

Controle de Fluxo

- O comando IF-ELSE
- ✓ Sabemos que o controle de fluxo é fundamental em qualquer linguagem de programação;
- ✓ Utilizamos comandos condicionais para controlar o fluxo que nosso programa deve seguir;
- ✓ O comando IF é comum em muitas linguagens de programação, sua função é verificar se uma condição é verdade ou falsa.
- Sintaxe:

Controle de Fluxo

➤ Sintaxe:

If(<expressão booleana ou variável booleana>){ } Este é um exemplo de uma sintaxe de if simples pode ser “traduzido” para: **se**(<expressão booleana ou variável booleana> = verdadeiro) { faça isso } Muitas vezes precisamos que algo seja executado quando a expressão avaliada não for verdadeira para isso utilizamos

o **else**.

Controle de Fluxo

➤ Sintaxe:

```
If (<expressão booleana ou variável booleana>){  
  }else{  
  }
```

Este é um exemplo de uma sintaxe de if pode ser

“traduzido” para:

```
se(<expressão booleana ou variável booleana> =  
verdadeiro) {  
  faça isso  
}senão{  
  faça isso  
}
```

Controle de Fluxo

➤ Forma geral em JAVA:

```
...  
    if(<expressão booleana>  
        <instrução 1>;  
        <próxima instrução>;
```

➤ Mecanismo:

- ✓ a <expressão booleana> é avaliada;
- ✓ se o valor lógico encontrado for true, a <instrução 1> será executada e depois o fluxo segue para a <próxima instrução>;
- ✓ se o valor lógico encontrado for false, a <instrução 1> não será executada e o fluxo segue para a próxima instrução.

Controle de Fluxo

```
int i = 10;
if (i > 10) {
    if (i == 10) {
        System.out.println("x igual a 10");
    }
    else if (i == i++) {
        System.out.println("i menor que 10");
    }
    else {
        System.out.println("i não é menor que 10");
    }
}
```

- ✓ A primeira condição é FALSE e não tem nenhum ELSE que gere um resultado.
- ✓ Veja todos os ELSE que existe é referente ao segundo IF.

Controle de Fluxo

➤ Tipo de dados booleano (*lógico*):

✓ Elementos:

- os valores lógicos: *true* e *false* (verdadeiro e falso).

Operações:

- *NÃO* (!) negação ;
- *E* (&&) conjunção ;
- *OU* (||) disjunção.

Definição das operações lógicas pela tabela verdade

não

p	$\sim p$
V	F
F	V

E

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

OU

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Controle de Fluxo

Exemplos de expressões lógicas:

$(a + x < 0)$ ou $(b = 0)$

$(a + x < 0) \parallel (b == 0)$

$(a > 0)$ e $(b - 6 \geq a)$

$(a > 0) \&\& (b - 6 \geq a)$

não($(a = 0)$ ou $(a = 1)$)

$! ((a == 0) \parallel (a == 1))$

Controle de Fluxo

➤ SWITCH

- ✓ A instrução switch por vezes chamada de switch...case possibilita a execução condicional de instruções de acordo com a correspondência entre a expressão avaliada e a constante em case. o switch só poderá ser aplicado
- ✓ a variáveis int, short, byte e char;
- ✓ Portanto qualquer tentativa de usar com variáveis boolean, long, double, flout e um objeto resultará em um erro de compilação.

```
intx = 3;  
switch (x) {  
  Case 1 : {  
    System.out.println("x vale 1");  
    break;  
  }  
  case 2:  
    System.out.println("x vale 2");  
    break;  
  default:  
    System.out.println("x é maior que 2");  
    break;  
}
```


Controle de Fluxo

Obreak:

- ✓ Todas às vezes em que precisarmos interromper um laço de repetição, podemos utilizar a instrução break;

Break, do inglês, significa "quebrar", "parar", e aqui na programação, o que a instrução faz é parar a execução ou então, interromper um laço num determinado momento. A seguir, temos um exemplo do uso da instrução **break**.

```
for(int i=0;i!=10;i++){  
  if(true) //quando uma condição for atendida  
    break;  
}
```

Controle de Fluxo

- ✓ O break é necessário, pois caso seja omitido, o fluxo seguirá normalmente como você pode ver no código seguinte.

```
int x = 2;  
switch (x) { case 1 : {  
System.out.println("x vale 1");  
System.out.println("x vale 1"); break; }  
case 2: System.out.println("x vale 2");  
break; default: System.out.println("x é  
maior que 2"); System.out.println("x é  
maior que 2"); break; }
```

R=

x vale 2

x é maior que 2

x é maior que 2

Controle de Fluxo

case:

- ✓ O condicional switch testa o valor contido em uma variável e o compara com os valores fornecidos em cada caso, representados pela palavra reservada **case**; Podemos ter quantos casos forem necessários na
- ✓ estrutura, e quando um dos valores corresponder ao da variável de teste, o código do caso será executado.

Control de Fluxo

```
public static void main(String[] args) {  
    Scanner entrada = new Scanner(System.in);  
    System.out.println("Entre com um número entre 1 e 4:");  
    int num = entrada.nextInt();  
    switch (num) {  
    case 1:  
        System.out.println("Você escolheu 1");  
        break;  
    case 2:  
        System.out.println("Você escolheu 2");  
        break;  
    case 3:  
        System.out.println("Você escolheu 3");  
        break;  
    case 4:  
        System.out.println("Você escolheu 4");  
        break;  
    default:  
        System.out.println("Número inválido");  
    }  
}
```

Controle de Fluxo

➤ WHILE –DO WHILE

❖ While:

- ✓ O termo **while** pode ser “enquanto” traduzido para o português como
Este termo é utilizado para construir uma estrutura de repetição
- ✓ que executa, repetidamente, uma única instrução ou um bloco delas “enquanto” uma expressão booleana for verdadeira.

```
public class ExemploWhile {  
    public static void main(String args[]) {  
        int contador = 0;  
        while(contador < 50) {  
            System.out.println("Repetição nr: " + contador);  
            contador++;  
        }  
    }  
}
```

Controle de Fluxo

- ✓ Nesse exemplo podemos ver, existe a variável “X” que é iniciada valendo 15, a cada loop executado (repetição) é somado 1 ao contador. Perceba que o while irá manter a repetição enquanto a variável “X” for menor que 18.

```
1 public class Tableless {  
2  
3 public static void main(String[] args) {  
4  
5     int x = 15;  
6  
7     while (x < 18) {  
8         System.out.println("Você não tem permissão para entrar");  
9         x++;  
10    }  
11  
12    }  
13 }
```

Controle de Fluxo

❖ Do-while:

- ✓ A estrutura de repetição do-while é uma variação da estrutura while. Existe uma diferença útil, porém importante, entre elas.
- ✓ Em um laço while, a condição é testada antes da primeira execução das instruções que compõem seu corpo.
- ✓ Se a condição for falsa na primeira vez em que for avaliada, as instruções desse laço não serão executadas nenhuma vez.

Controle de Fluxo

- ✓ Em um laço do-while, por outro lado, a condição somente é avaliada depois que suas instruções são executadas pela primeira vez, mesmo que a condição desse laço seja falsa antes de ele iniciar, suas instruções serão executadas pelo menos uma vez.

```
1 public class Tableless {  
2  
3     public static void main(String[] args) {  
4  
5         int x = 15;  
6  
7         do {  
8  
9             System.out.println("Você não tem permissão para entrar");  
10            x++;  
11  
12        } while (x < 18);  
13    }  
14 }
```


Controle de Fluxo

➤ Estrutura For

✓
✓ O laço for é uma estrutura de repetição compacta.

Seus elementos de inicialização, condição e iteração são reunidos na forma de um cabeçalho e o corpo é disposto em seguida.

Controle de Fluxo

- ✓ O comando FOR é utilizado quando você quantas vezes precisa fazer a interação. Ele é composto por três expressões, sendo elas:

- a) Declaração e inicialização
- b) Expressão condicional
- c) Expressão de interação

```
1  for ([expressão inicial]; [a expressão condicional]; [atualização da expressão inicial]) {  
2      // aqui entra o código a ser repetido n vezes  
3  }
```

```
1  public class Tableless {  
2  
3      public static void main(String[] args) {  
4  
5          for(int x = 0; x<1000; x++){  
6              System.out.println("Valor: " + x);  
7          }  
8      }  
9  }
```

Controle de Fluxo

✓ Quando o comando "for" é executado, a primeira coisa a ser feita é inicialização prevista. Em seguida, verifica-se se CONDIÇÃO é

"falso". Em caso afirmativo, o loop não será executado. Caso contrário, todos os comandos existentes no bloco abaixo do comando são executados e a operação prevista em EXPRESSÃO é executada. Mais uma vez, a CONDIÇÃO é analisada. Caso ela seja falsa, o loop é interrompido. Caso contrário, ela continua.

```
for(int i=1;i<=10;i++)           //O loop é executado 10 vezes
    System.out.println(i+" "); //Será impressa na tela uma contagem de 1 até 10.
```

VETORES EM JAVA

- ✓ Vetores são estruturas de dados que armazenam usualmente uma quantidade fixa de dados de um certo tipo; por esta razão, também são
- ✓ conhecidos como estruturas homogêneas de dados;
Internamente, um vetor armazena diversos valores, cada um
- ✓ associado a um número que se refere à posição de armazenamento, e é conhecido como índice;
- ✓ Os vetores são estruturas indexadas, em que cada valor que pode ser armazenado em uma certa posição (índice) é chamado de elemento do vetor.

VETORES EM JAVA

- ✓ Cada posição de um vetor é unicamente identificada por um valor inteiro positivo, linear e sequencialmente numerado;
- ✓ Abaixo é mostrado se dá esse acesso aos seus elementos, lembrando que sempre sua numeração começa em 0.

Um VETORde 5 elementos

3	41	28	79	7
0	1	2	3	4

Fonte: <https://www.devmedia.com.br>

VETORES EM JAVA

❖ Declarando Variáveis do Tipo Vetor

✓ deverão ser fornecidas as seguintes informações:

✓ 1) o nome do vetor;

✓ 2) o número de posições do vetor (seu tamanho);

✓ 3) o tipo de dado que será armazenado no vetor;

✓ A declaração de um vetor para "inteiros", denome "vetor", em

Java:

Fonte: <https://www.devmedia.com.br>

```
1 | int vetor[]; // declaração do vetor
```

✓ Embora declarado, o vetor não está pronto para uso, sendo necessário reservar

✓ espaço para seus elementos (uma operação de alocação de memória).

Na alocação de espaço, não repetimos os colchetes e utilizamos o operador new

```
1 | vetor = new int[10]; // alocação de espaço para vetor
```

Fonte: <https://www.devmedia.com.br>

✓ As duas declarações podem ser combinadas em uma única, mais compacta.

```
1 | int vetor[] = new int[10]; // declaração combinada
```

Fonte: <https://www.devmedia.com.br>

VETORES EM JAVA

```
1  int n = 10; // tamanho do vetor
2  int v[] = new int[n]; // declaração e alocação de espaço para o vetor "v"
3  int i; // índice ou posição
4
5  // processando os "n" elementos do vetor "v"
6  for (i=0; i<n; i++) {
7      v[i] = i; // na i-ésima posição do vetor "v" armazena o valor da variável "i"
8  }
```

Fonte: <https://www.devmedia.com.br>

✓ Representação interna:

v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]
0	1	2	3	4	5	6	7	8	9

Fonte: <https://www.devmedia.com.br>

VETORES EM JAVA

```
public class VetorUm {  
    public static void main(String[] args) {  
        int v[ ] = {1,2,3,4,5};  
        System.out.println("Total de casas de v é: " +  
v.length);  
        for(int i = 0; i <= 4; i++) {  
            System.out.println( "Na posição " + i + " temos o  
valor " + v[i] );  
        }  
    }  
}
```


<https://www.devmedia.com.br/vetores-em-java/21449>

